```
subroutine foo (A,B,C,M,N)
double precision A(*), B(*), C(*), SUM1
integer M, N, I, J
call samp_auxiliary_routine(M,N)
sum1=0,d0
do I = 1, M
  sum1 = sum1 + A(I)
    do J = 1, N
         sum1 = sum1 + A((I-1)*M+J) + B((I-1)*M+J)
    enddo
    C(I) = sum1
enddo
return
end
```

Figure 1

```
samp_auxiliary_routine_(A,B)
int *A, *B;
{ *A = ARRAY_SIZE;
  *B = ARRAY_SIZE;
}
```

Figure 2

```
supercompiler_timer()
{
 extern void (*supercompiler_routine)()

 /* Example usage actually calls the target subroutine
twice, so time that:*/

 (*supercompiler_routine)(A, B, C, &N, &M);
 (*supercompiler_routine)(A, B, C, &N, &M);
}
```

Figure 3

```
#define ARRAY_SIZE 1024
int N=0
intM=0
double A[ARRAY_SIZE*ARRAY_SIZE];
double B[ARRAY_SIZE*ARRAY_SIZE];
double C[ARRAY_SIZE];

supercompiler_initialize()
{
 int i;

 /* Initialize the variables: */
 for (i = 0; i<ARRAY_SIZE*ARRAY_SIZE ; i++)
 {
  A[i] = (double) i;
  B[i] = (double) i;
 }
 for ( i = 0 ; i < ARRAY_SIZE ; i++ ) C[i] = (double) i;

 /* The target subroutine needs to call an auxiliary
routine: */

supercompiler_binding("samp_auxiliary_routine_",&samp
_auxiliary_routine_);

 /* Place the caches into a known state: clean */
 flush_cache();
}
```

Figure 4

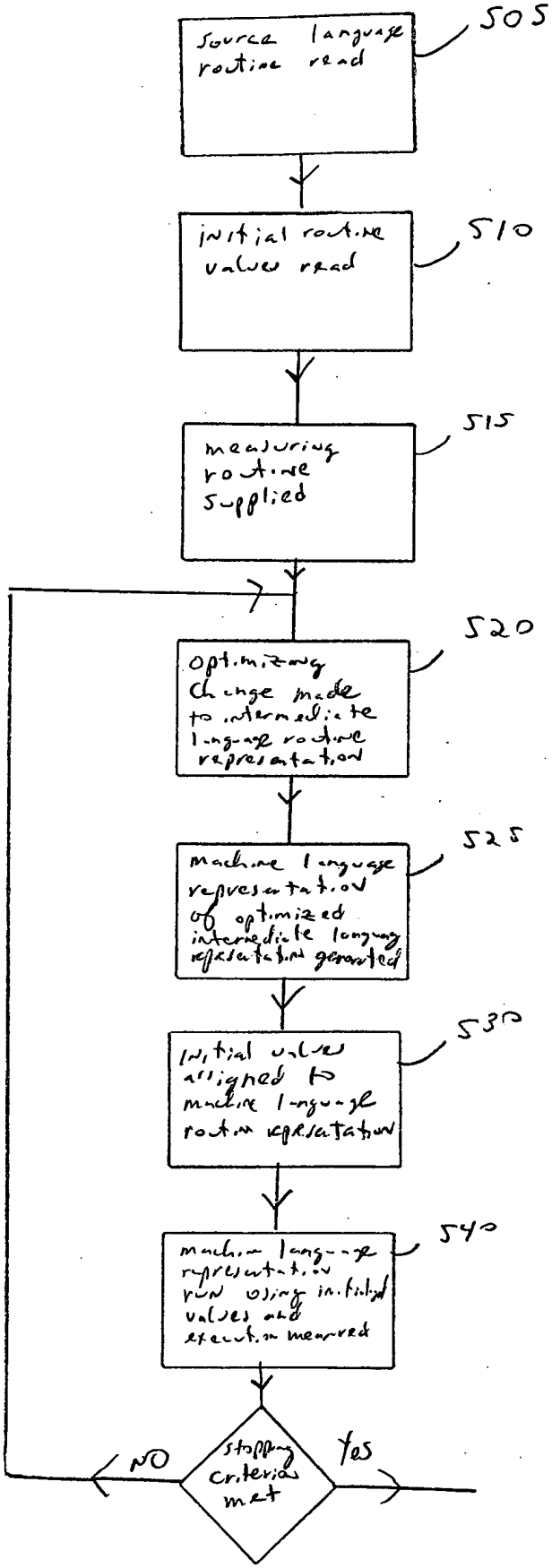
source language routine read
505
initial routine values read
510
measuring routine supplied
515
optimizing change made to intermediate language routine representation
520
machine language representation of optimized intermediate language representation generated
525
initial values assigned to machine language routine representation
530
machine language representation run using initial values and execution measured
540
stopping criteria met
NO
YES

FIGURE 5

Interactive menu:
625 1.) Change the location of a task
630 2.) Move all of one variable one step up
635 3.) Move all of one variable one step down
610 4.) Display tasks in a range (%d to %d)
5.) Search for a given load value
6.) Change the order of a sequence
7.) Reset to the original ordering
620 8.) Time an ordering
615 9.) Generate assembly code for an ordering
10.) Generate greg-code for an ordering
11.) Display tasks that cause a stack spill
12.) Display interaction permutation to date
13.) Remove all dummy assigned labels
14.) Add/Remove optional fxchs to all additions
15.) Add/Remove one optional fxchs to a task
16.) Leave the interactive menu

Figure 6

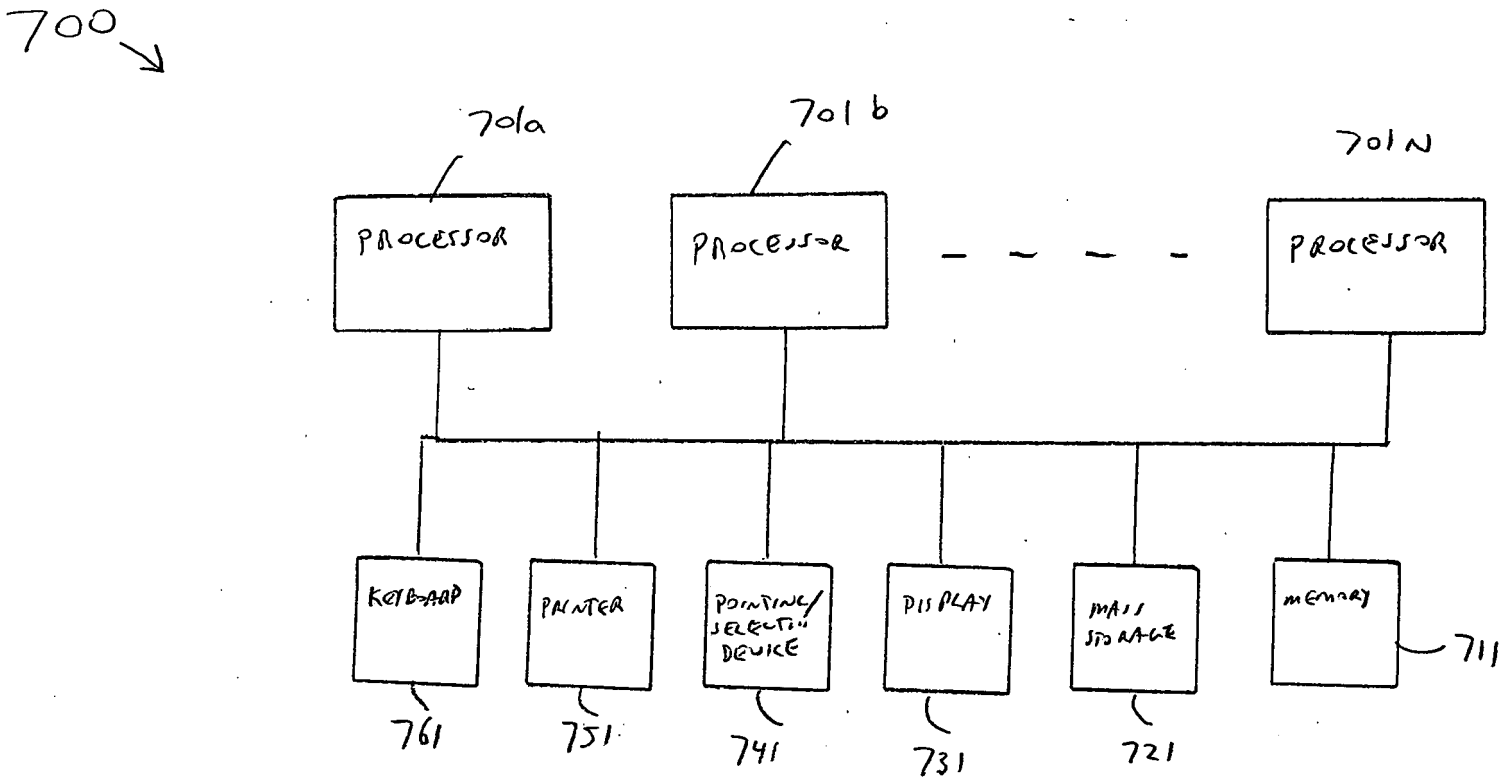
700
701a
701b
701N
PROCESSOR
PROCESSOR
PROCESSOR
KEYBOARD
PRINTER
POINTING/ SELECTION DEVICE
DISPLAY
MASS STORAGE
MEMORY
711
761
751
741
731
721

FIGURE 7